Data Compression
Lecture Notes

Dr. Faisal Aslam

Lecture 1 Introduction to Data Compression

1.1 Learning Objectives

By the end of this lecture, students will be able to:

- Understand the motivation and benefits of data compression
- Differentiate between lossless and lossy compression techniques
- Compute and interpret common compression performance metrics
- Apply the Huffman coding algorithm step by step
- Analyze real-world compression trade-offs

1.2 Introduction and Motivation: Why Compress Data?

Data compression is the process of representing information using fewer bits than its original form. It is a fundamental component of modern computing systems, enabling efficient storage, faster communication, and reduced operational costs.

Everyday applications of compression include:

- Streaming audio and video
- Image storage and sharing
- File archiving and backups
- Network communication and cloud services

Definition

Data Compression is the process of reducing the number of bits required to represent information, either:

- **Losslessly**: allowing exact reconstruction of the original data
- **Lossily**: allowing controlled loss of information to achieve higher compression

1.2.1 Benefits of Data Compression

Data compression provides three key benefits that are critical in modern computing:

1. **Reduce Storage Space**:
    - Allows more data to be stored in the same physical space
    - Enables archival of historical data that would otherwise be discarded
    - Reduces hardware requirements for storage systems
2. **Reduce Communication Time and Bandwidth**:
    - Enables faster file transfers and downloads
    - Makes high-quality streaming (4K/8K video) practical over limited bandwidth
    - Reduces latency in real-time applications like video conferencing and online gaming
    - Allows IoT devices to transmit data efficiently over wireless networks
3. **Save Money**:
    - Reduces cloud hosting costs (storage and egress fees)
    - Lowers communication costs for data transmission
    - Decreases capital expenditure on storage hardware
    - Reduces energy consumption for data centers and network infrastructure

1.3 Lossless vs. Lossy Compression

1.3.1 Lossless Compression

Lossless compression guarantees perfect reconstruction of the original data. It is essential when accuracy and data integrity are critical.

Typical applications:

- Text files and source code
- Executables and databases
- Medical, scientific, and legal data

1.3.2 Lossy Compression

Lossy compression achieves higher compression ratios by discarding information that is less perceptible or less important.

Typical applications:

- Audio (MP3, AAC)
- Images (JPEG)
- Video (H.264, H.265)

1.3.3 Choosing Between Lossless and Lossy

Factor	**Lossless Compression**	**Lossy Compression**
Reconstruction	Exact	Approximate
Data sensitivity	High	Moderate to low
Typical ratios	Low to moderate	High
Quality impact	None	Controlled degradation

Table 1: Lossless vs. Lossy Compression

1.4 Compression Performance Metrics

1.4.1 Size-Based Metrics

$$\text{Compression Ratio (CR)} = \frac{\text{Original Size}}{\text{Compressed Size}}$$
$$\text{Compression Factor} = \frac{\text{Compressed Size}}{\text{Original Size}}$$
$$\text{Space Savings (\%)} = \left(1 - \frac{\text{Compressed Size}}{\text{Original Size}}\right) \times 100\%$$

Interpretation:

- Larger compression ratios indicate better compression
- Smaller compression factors indicate better compression

1.4.2 Rate-Based Metrics

$$\text{Bits per Sample (bps)} = \frac{\text{Compressed Size (bits)}}{\text{Number of samples}}$$
$$\text{Bit-rate (bps)} = \frac{\text{Compressed Size (bits)}}{\text{Time (seconds)}}$$

These metrics are particularly important in audio and video compression systems.

1.5 Worked Example: Audio Compression Metrics

Example

Uncompressed Audio Properties

- Duration: 180 seconds
- Sampling rate: 44.1 kHz
- Bit depth: 16 bits
- Channels: 2 (stereo)

Original Size Calculation

$$\text{Total samples} = 180 \times 44{,}100 \times 2 = 15{,}876{,}000$$
$$\text{Size (bits)} = 15{,}876{,}000 \times 16 = 254{,}016{,}000$$
$$\text{Size (MB)} = \frac{254{,}016{,}000}{8 \times 1{,}048{,}576} \approx 30.27$$

Compression Results

Method	Size (MB)	CR	Savings	Bit-rate
FLAC (lossless)	18.16	1.67:1	40%	807 kbps
MP3 @ 320 kbps	6.75	4.49:1	77.7%	320 kbps
AAC @ 256 kbps	5.40	5.61:1	82.2%	256 kbps

1.6 Huffman Coding

Huffman coding is a widely used **lossless compression algorithm** that assigns variable-length binary codes to symbols based on their frequencies. More frequent symbols receive shorter codes.

1.6.1 Step-by-Step Huffman Coding Example

Example

Message: `MISSISSIPPI RIVER` (17 characters including space)
Symbol Frequencies

Symbol	Frequency
I	5
S	4
P	2
R	2
M	1
V	1
E	1
(space)	1

Tree Construction

1. Combine M(1) + V(1) $\to$ 2
2. Combine E(1) + (space)(1) $\to$ 2
3. Combine P(2) + R(2) $\to$ 4
4. Combine 2 + 2 $\to$ 4
5. Combine 4 + 4 $\to$ 8
6. Combine I(5) + S(4) $\to$ 9
7. Combine 8 + 9 $\to$ 17

One Possible Code Assignment

Symbol	Code	Length
I	00	2
S	01	2
P	100	3
R	101	3
M	1100	4
V	1101	4
E	1110	4
(space)	1111	4

Compressed Size

$$5(2) + 4(2) + 2(3) + 2(3) + 4(1) = 52 \text{ bits}$$

Original Size (ASCII) $= 17 \times 8 = 136$ bits
Compression Ratio $= 136/52 \approx 2.62 : 1$

1.6.2 Key Properties of Huffman Coding

- Produces prefix-free codes
- Enables instantaneous decoding
- Guarantees minimum average code length among prefix codes
- Widely used in practical compression systems

1.7 End of Chapter Questions

Exercise Lecture 1.0

Problem 1: Basic Compression Metrics

An uncompressed grayscale image has the following properties:

- Resolution: 1024×1024 pixels
- Bit depth: 8 bits per pixel

After compression, the image size is 320 KB.
Calculate:

(a) Original image size in KB

(b) Compression ratio

(c) Compression factor

(d) Space savings percentage

Exercise Lecture 1.1

Problem 2: Audio Bit-rate and Storage

A mono audio recording has the following parameters:

- Duration: 5 minutes
- Sampling rate: 48 kHz
- Bit depth: 16 bits

The file is compressed using a lossy codec to a constant bit-rate of 192 kbps.
Calculate:

(a) Size of the uncompressed audio file in MB

(b) Size of the compressed file in MB

(c) Compression ratio

(d) Bits per sample after compression

Exercise Lecture 1.2

Problem 3: Comparing Compression Options

A video clip has an uncompressed data rate of 120 Mbps. Three compression options are available:

Option	Compressed Bit-rate
A	6 Mbps
B	3 Mbps
C	1.5 Mbps

For each option, calculate:

(a) Compression ratio

(b) Data consumed for a 10-minute video (in MB)

Which option would you choose for:

(i) Live video streaming?

(ii) Archival storage?

Briefly justify your answers.

Exercise Lecture 1.3

Problem 4: Huffman Coding Construction

Given the following symbol frequencies:

Symbol	Frequency
A	10
B	8
C	6
D	5
E	4
F	3
G	2
H	2

(a) Construct the Huffman tree step by step

(b) Assign a binary code to each symbol

(c) Compute the total number of bits required to encode the message

(d) Calculate the average number of bits per symbol

Exercise Lecture 1.4

Problem 5: Fixed-Length vs. Huffman Coding
Using the symbol set from Problem 4:

(a) Determine the minimum fixed-length code required

(b) Compute the total number of bits using fixed-length coding

(c) Compare the result with Huffman coding

(d) Calculate the percentage reduction in total bits achieved by Huffman coding

Exercise Lecture 1.5

Problem 6: Text Compression Scenario
A text file contains 50,000 characters and is stored using 8-bit ASCII encoding. After compression using a lossless algorithm, the file size becomes 18 KB. Calculate:

(a) Original file size in KB

(b) Compression ratio

(c) Compression factor

(d) Space savings percentage

Explain why compression ratios for text files vary significantly depending on content.

Exercise Lecture 1.6

Problem 7: Practical Design Question
You are designing a compression system for a wearable health-monitoring device that:

- Records sensor data continuously
- Has limited storage capacity
- Requires exact data reconstruction
- Operates on a low-power processor

(a) Should the system use lossless or lossy compression? Explain.

(b) Which performance metrics are most important in this scenario?

(c) Would a variable-length coding scheme be appropriate? Why or why not?

Lecture 2 Theory of Compression — Limits and Optimality

2.1 Basic Terminology and Notation

Definition

Alphabet

An *alphabet* $\mathcal{X}$ is a finite set of possible symbols. Examples:

- Binary alphabet: $\mathcal{X} = \{0, 1\}$
- English letters: $\mathcal{X} = \{\mathtt{A}, \ldots, \mathtt{Z}\}$
- Bytes: $\mathcal{X} = \{0, 1, \ldots, 255\}$

Definition

Symbol

A *symbol* is a single element drawn from an alphabet. For example, the letter `E` is a symbol from the English alphabet.

Definition

Random Variable

A *random variable* X is a mathematical model of a source that produces symbols. It assigns probabilities to symbols in the alphabet:

$$P(X = x), \quad x \in \mathcal{X}$$

Entropy is defined on random variables, not directly on symbols themselves.

Definition

Source

A *source* is a process that generates a sequence of symbols $(X_1, X_2, X_3, \ldots)$ according to some probability law. In this lecture, we assume discrete sources unless stated otherwise.

Definition

Message (or Sequence)

A *message* is a finite sequence of symbols generated by the source:

$$x^n = (x_1, x_2, \ldots, x_n)$$

Compression algorithms operate on messages, not on individual symbols.

Definition

Code and Codewords

A *code* assigns a binary string (codeword) to each symbol or message.

- Source symbols → codewords (e.g., Huffman coding)
- Messages → bitstreams (e.g., arithmetic coding)

Definition

Block Length

The *block length* n is the number of source symbols grouped together and encoded as a unit. Larger block lengths generally allow better compression but increase delay and complexity.

Definition

Model

A *model* estimates the probabilities of symbols or sequences. Better models lead to better compression by reducing uncertainty.

2.2 Information and Redundancy: The Core Concepts

2.2.1 Information: A Formal Measure of Uncertainty Reduction

In information theory, information is defined rigorously as a **quantitative measure of the reduction in uncertainty** that results from observing the outcome of a random event.

Definition Lecture 2.1. *Let X be a random event that occurs with probability $p = \Pr(X)$. The **information content** (or* self-information*) $I(X)$ provided by the occurrence of X is defined as:*

$$I(X) = \log_b \left(\frac{1}{p}\right) = -\log_b(p)$$

where:

- $b = 2$ *yields **bits** (binary digits)*
- $b = e$ *yields **nats** (natural units)*
- $b = 10$ *yields **hartleys** or **dits***

Example

Predictability vs. Information:

- In a city where it rains every day, the statement "It rained today" conveys almost no information because it was expected
- A file that contains only the bit '1' provides very little information
- A coin that always lands heads produces outcomes, but no information

  Key idea: Perfect predictability implies zero information gain.

Example

Daily Weather Forecast — Information Content:

- Sunny in Phoenix (probability 0.9): $I = -\log_2 0.9 \approx 0.15$ bits
- Snow in Phoenix (probability 0.001): $I = -\log_2 0.001 \approx 9.97$ bits
- Rain in Seattle (probability 0.3): $I = -\log_2 0.3 \approx 1.74$ bits

Interpretation: Rare events carry more information because they reduce uncertainty the most.

2.2.2 Redundancy: The Enemy of Information and the Friend of Compression

Redundancy refers to predictable or repeated structure in data. It is what allows data to be represented using fewer bits.

1. **Spatial Redundancy**: Neighboring data values are highly correlated

   Example

   In a photograph of a clear blue sky, most neighboring pixels have nearly identical color values.

   - **Naive**: Store the RGB value of each pixel independently
   - **Smarter**: Encode repeated pixel values using run-length encoding
   - **Even smarter**: Predict each pixel from its neighbors and encode only the small prediction error

2. **Statistical Redundancy**: Some symbols occur far more frequently than others

Example

English letter frequencies:

Letter	Frequency	Letter	Frequency
E	12.7%	Z	0.07%
T	9.1%	Q	0.10%
A	8.2%	J	0.15%

Frequent letters get shorter codes in variable-length coding schemes.

3. **Knowledge Redundancy**: Information already known to both encoder and decoder

4. **Perceptual Redundancy**: Information that humans cannot perceive

2.3 Entropy: The Fundamental Limit

2.3.1 What is Entropy? Different Perspectives

Definition

Shannon Entropy of a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$ having probabilities $\{p_1, p_2, \ldots, p_n\}$:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \quad \text{bits}$$

Two Complementary Interpretations:

1. **Average Information Content**: Expected value of information content across all symbols

2. **Uncertainty or Surprise**: Measures how uncertain we are about the next symbol

2.3.2 Calculating Entropy: Step by Step

Example

Binary Source Example - Detailed Calculation:
Consider a biased coin: P(Heads) = 0.8, P(Tails) = 0.2

Step 1: **Calculate individual information content**:

$$I_H = -\log_2(0.8) \approx 0.3219 \text{ bits}$$
$$I_T = -\log_2(0.2) \approx 2.3219 \text{ bits}$$

Step 2: **Calculate entropy as expected value**:

$$H = 0.8 \times 0.3219 + 0.2 \times 2.3219 = 0.7219 \text{ bits}$$

Step 3: **Verify using direct formula**:

$$H = -[0.8 \log_2(0.8) + 0.2 \log_2(0.2)] \approx 0.7219 \text{ bits}$$

Key Insights:

- **Extreme cases**:
  - Fair coin (P=0.5): $H = 1.0$ bit (maximum uncertainty)
  - Always heads (P=1.0): $H = 0$ bits (no uncertainty)
  - 90% heads: $H \approx 0.469$ bits

2.3.3 Entropy of English Text: A Practical Case Study

Example

Calculating English Letter Entropy:
Based on letter frequencies in typical English text:

$$H \approx 4.18 \text{ bits/letter}$$

Layered Interpretation:

- **First-order entropy (letters independent)**: 4.18 bits/letter
- **Actual uncertainty is lower**: Letters have dependencies (Q→U)
- **Comparison with encoding schemes**:

Encoding Method	**Bits/Letter**
Naive (5 bits for 26 letters)	5.00
Huffman (letter-based)	4.30
Using digram frequencies	3.90
Using word frequencies	2.30
Optimal with full context	∼1.50

2.3.4 Beyond First-Order Entropy: The Full Picture

Higher-Order Entropies quantify uncertainty while accounting for increasing context:

- **Zero-order entropy** (H_0): $H_0 = \log_2 |\mathcal{X}|$
- **First-order entropy** (H_1): $H_1 = -\sum p(x) \log_2 p(x)$
- **Second-order entropy** (H_2): $H_2 = -\sum p(x, y) \log_2 p(x|y)$
- N**th-order entropy** (H_N): $H_N = -\sum p(x_1, \ldots, x_N) \log_2 p(x_N \mid x_1, \ldots, x_{N-1})$

2.3.5 Entropy Rate

The **entropy rate** of a source is defined as the limiting uncertainty per symbol when arbitrarily long contexts are available:

$$H_\infty = \lim_{N \to \infty} H_N$$

2.3.6 The Entropy Theorem: Why It Matters

Important

Shannon's Source Coding Theorem
Let a discrete memoryless source have entropy H.

1. **Converse (Impossibility)**: No lossless coding scheme can achieve $L < H$
2. **Achievability (Possibility)**: For any $\epsilon > 0$, there exists a coding scheme with $H \leq L < H + \epsilon$ for sufficiently large block sizes

2.3.7 Key Takeaways

- Entropy measures both **average information** and **uncertainty**
- Higher-order models reduce entropy by exploiting dependencies
- The entropy rate represents the ultimate compression limit
- Shannon's theorem precisely separates the *possible* from the *impossible*

2.4 Entropy as a Lower Bound

2.4.1 The Fundamental Inequality

Theorem Lecture 2.2 (Entropy Lower Bound)**.** *For any* ***uniquely decodable*** *code C for source X:*

$$L(C) \geq H(X)$$

where $L(C) = \mathbb{E}[\ell(X)] = \sum_i p_i \ell_i$ is the expected code length.

Example

Binary Source with $p(0) = 0.9$, $p(1) = 0.1$

$$H = -0.9 \log_2 0.9 - 0.1 \log_2 0.1 \approx 0.469 \text{ bits/symbol}$$

Why we can't achieve 0.4 bits/symbol:

1. For 100 symbols, typical sequences: $2^{100 \times 0.469} \approx 2^{46.9}$
2. To encode all uniquely, need at least $2^{46.9}$ codewords
3. At 0.4 bits/symbol, total bits = 40
4. \# codewords $\leq 2^{40} < 2^{46.9} \rightarrow$ impossible!

2.5 Kraft-McMillan Inequality: Core Theoretical Tool

2.5.1 Statement and Interpretation

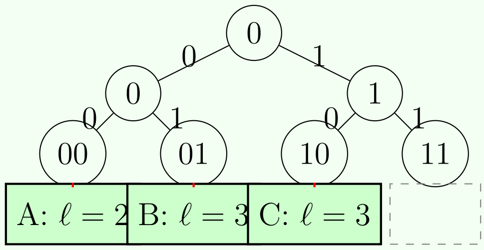
Theorem Lecture 2.3 (Kraft-McMillan Inequality (Binary Case))**.** *Let $\ell_1, \ell_2, \ldots, \ell_m$ be the lengths of codewords in a **prefix code**. Then:*

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$$

Conversely, if integers $\ell_1, \ldots, \ell_m$ satisfy this inequality, then there exists a binary prefix code with these lengths.

Example

Tree Visualization of Kraft Inequality
Consider a binary tree of depth $L = \max \ell_i$:



Calculating Kraft sum for $\{\ell_A = 2, \ell_B = 3, \ell_C = 3\}$:

$$\sum 2^{-\ell_i} = 2^{-2} + 2^{-3} + 2^{-3} = 0.25 + 0.125 + 0.125 = 0.5 \leq 1$$

Example

Testing Code Feasibility

1. **Valid lengths**: $\{1, 2, 3, 3\}$

$$\sum = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 0.5 + 0.25 + 0.125 + 0.125 = 1.0 \quad \text{VALID}$$

2. **Invalid lengths**: $\{1, 1, 2\}$

$$\sum = 2^{-1} + 2^{-1} + 2^{-2} = 0.5 + 0.5 + 0.25 = 1.25 > 1 \quad \text{INVALID}$$

2.6 Optimality of Huffman Codes (Theory Only)

2.6.1 The Optimality Theorem

Theorem Lecture 2.4 (Huffman Optimality)**.** *Given a source with symbol probabilities* $p_1, p_2, \ldots, p_m$*, the Huffman algorithm produces a prefix code that* ***minimizes*** *the expected code length* $L = \sum_{i=1}^{m} p_i \ell_i$ *among all prefix codes.*

2.6.2 Relation to Entropy

For any Huffman code:

$$H(X) \leq L_{\text{Huffman}} < H(X) + 1$$

Example

Understanding the "+1" Gap
Consider source with probabilities $\{0.6, 0.3, 0.1\}$:

$$H \approx 1.295 \text{ bits}$$

Ideal (non-integer) lengths: $-\log_2 p_i = \{0.737, 1.737, 3.322\}$
Huffman code: 0.6→0, 0.3→10, 0.1→11

$$L = 0.6 \times 1 + 0.3 \times 2 + 0.1 \times 2 = 1.4 \text{ bits}$$

Comparison:

- Entropy: 1.295 bits
- Huffman: 1.400 bits
- Gap: 0.105 bits (much less than 1!)

2.6.3 Why Huffman is Optimal but Not Perfect

Important

Huffman is Optimal Within a Restricted Class

Huffman is optimal among:

- **Symbol-by-symbol** codes
- **Prefix** codes
- **Static** codes

But real optimality might require:

- **Block coding**
- **Fractional bits** (arithmetic coding)
- **Adaptive probabilities**

2.7 Limitations of Symbol-by-Symbol Coding

2.7.1 Three Fundamental Limitations

1. **Cannot Exploit Dependencies**
2. **Integer Length Constraint**: $\ell_i \in \mathbb{Z}^+$ but $-\log_2 p_i \in \mathbb{R}$
3. **Memoryless Assumption**

Example

English Text: The Cost of Symbol-by-Symbol

- **First-order entropy** (ignoring dependencies): 4.0 bits/letter
- **Actual entropy rate** (with dependencies): 1.5 bits/letter
- **Huffman on letters**: 4.0 bits/letter
- **Gap**: 2.5 bits/letter wasted due to ignoring dependencies

2.8 Block Coding and Improved Efficiency

2.8.1 The Block Coding Idea

Instead of coding symbols individually, group them into blocks of length n:

$$\mathbf{X} = (X_1, X_2, \ldots, X_n)$$

Definition

nth Extension of a Source
For a source with alphabet $\mathcal{X}$, the nth extension has alphabet:

$$\mathcal{X}^n = \{(x_1, \ldots, x_n) : x_i \in \mathcal{X}\}$$

with size $|\mathcal{X}|^n$.

2.8.2 Key Mathematical Results

Theorem Lecture 2.5 (Entropy of Block Source)**.** *For a discrete memoryless source:*

$$H(X^n) = nH(X)$$

Theorem Lecture 2.6 (Block Coding Performance)**.** *There exists a prefix code C_n for X^n such that:*

$$nH(X) \leq L_n < nH(X) + 1$$

Dividing by n:

$$H(X) \leq \frac{L_n}{n} < H(X) + \frac{1}{n}$$

Important

The Magic of Block Coding
As $n \to \infty$:

$$\frac{L_n}{n} \to H(X)$$

We can approach entropy **arbitrarily closely** by making blocks larger!

2.8.3 Step-by-Step Example

Example

Binary Source: $p(0) = 0.9$, $p(1) = 0.1$, $H \approx 0.469$

Step 1: $n = 1$ **(symbol-by-symbol)**

- Huffman: 0→0, 1→1
- $L_1 = 1$ bit/symbol
- Efficiency: $\eta = 0.469/1 = 46.9\%$

Step 2: $n = 2$ **(code pairs)**

- Block probabilities: $P(00) = 0.81$, $P(01) = 0.09$, $P(10) = 0.09$, $P(11) = 0.01$
- Codes: 00→0, 01→10, 10→110, 11→111
- Per symbol: $L_2/2 = 0.645$ bits/symbol
- Efficiency: $\eta = 0.469/0.645 = 72.7\%$

2.9 Trade-Offs in Block Coding

2.9.1 The Engineering Challenges

1. **Exponential Alphabet Growth**: $|\mathcal{X}^n| = |\mathcal{X}|^n$
2. **Memory Requirements**: Huffman tree has $2m^n - 1$ nodes
3. **Computational Complexity**: $O(m^n \log m^n)$
4. **Delay and Latency**: Must wait for n symbols

2.10 From Block Coding to Modern Compression

2.10.1 Arithmetic Coding: Fractional-Bit Block Coding

Important

Arithmetic Coding as "Infinite Block Coding"

Arithmetic coding cleverly avoids the exponential growth problem:

- **Idea**: Encode entire message as a single real number in [0,1)
- **No explicit blocks**: Processes symbols sequentially
- **Fractional bits**: Achieves $L \approx H(X)$ without large n

- **Removes integer constraint**: No "+1" overhead!

2.10.2 Context Modeling: Approximating Large Blocks

Instead of explicit block coding, modern compressors use:

1. **Context Models**: Predict next symbol based on previous k symbols
2. **Prediction + Residual Coding**: Encode only prediction error
3. **Dictionary Methods (LZ family)**: Build dictionary of previously seen phrases

2.11 What Theory Guarantees vs What Practice Achieves

Aspect	Theory Guarantees	Practice Achieves
Optimality	Can approach entropy arbitrarily closely	Gets close, but with practical limits
Block Size	$n \to \infty$ gives optimality	n limited by memory, latency, complexity
Complexity	Ignored, infinite resources allowed	Critical constraint; often dominates design

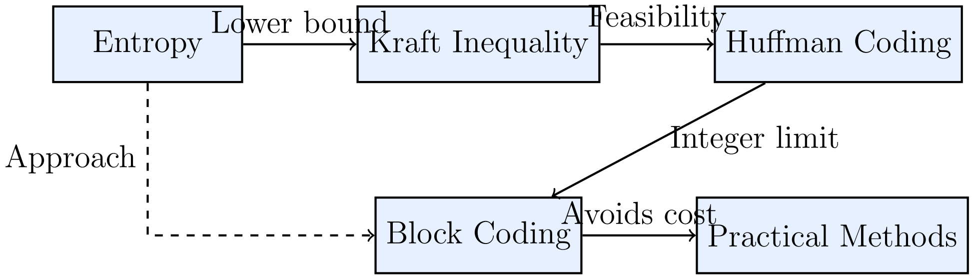
2.12 Summary and Key Takeaways

Important

Five Fundamental Lessons

1. **Entropy is the Absolute Limit**: $L \geq H(X)$ for any lossless code
2. **Kraft-McMillan Constrains All Codes**: $\sum 2^{-\ell_i} \leq 1$
3. **Huffman is Optimal Among Prefix Codes**: But limited by integer lengths
4. **Block Coding Allows Approaching Entropy**: $\lim_{n\to\infty} \frac{L_n}{n} = H(X)$
5. **Practical Compression Balances Efficiency and Complexity**

2.12.1 The Big Picture



2.12.2 Looking Forward

- **Next lecture: Arithmetic Coding**: Removes integer constraint
- **Then: Dictionary Methods (LZ family)**: Adaptive to data statistics
- **Finally: Modern Compressors**: Combining multiple techniques

Final Thought

Shannon's 1948 paper told us *exactly how good compression could possibly be.* Every compressor since has been trying to approach that limit while staying within practical constraints.

The gap between theory and practice is where engineering creativity lives!